```
//segment tree with lazy
typedef long long ll;
typedef long l;
#define floop(i,n) for(ll i=0;i<n;i++)
#define floopk(i,n,k) for(ll i=0;i<n;i+=k)
#define si(n) scanf("%lld",&n)
#define po(n) printf("%lld\n",n)
 ll  tree[500000];
 ll  lazy[500000];

/*void construct_tree(){
    for(l p=1;p<400000;p++){
        tree[p]=0;
        lazy[p]=0;
    }


}*/

ll query(ll index,ll trex,ll trey,ll qrex, ll qrey){

    ll mid=(trex+trey)/2;

    if(qrex>trey || qrey<trex)
        return 0;

    if(lazy[index]!=0){
        tree[index]+=lazy[index];
        if(trex!=trey){

            lazy[2*index]+=(lazy[index]/(trey-trex+1))*(mid-trex+1);
            lazy[2*index+1]+=(lazy[index]/(trey-trex+1))*(trey-mid);
        }

        lazy[index]=0;
    }



    if(qrex<=trex && qrey>=trey)
        return tree[index];

    return
query(2*index,trex,mid,qrex,qrey)+query(2*index+1,mid+1,trey,qrex,qrey);

}

void update(ll index,ll trex,ll trey,ll qrex,ll qrey,ll val){

    ll mid=(trex+trey)/2;


    if(lazy[index]!=0){
        tree[index]+=lazy[index];

        if(trex!=trey){

            lazy[2*index]+=(lazy[index]/(trey-trex+1))*(mid-trex+1);
            lazy[2*index+1]+=(lazy[index]/(trey-trex+1))*(trey-mid);
```

```
        }

        lazy[index]=0;
    }


    if(trex>trey || trex>qrey || trey<qrex)
        return;

    if(trex>=qrex && trey<=qrey){
        tree[index]+=(trey-trex+1)*val;
        if(trex!=trey){

            lazy[2*index]+=(mid-trex+1)*val;
            lazy[2*index+1]+=(trey-mid)*val;

        }
        return;

    }

    update(2*index,trex,mid,qrex,qrey,val);
    update(2*index+1,mid+1,trey,qrex,qrey,val);

    tree[index]=tree[2*index]+tree[2*index+1];

}

int main()  {
     //std::ios_base::sync_with_stdio(false);
    ll t;
    si(t);


    while(t--){

       //construct_tree();
       memset(tree,0,sizeof tree);
       memset(lazy,0,sizeof lazy);

        ll n;ll c;
        si(n);si(c);

        while(c--){
            ll tp;si(tp);

            if(tp==0){
                ll x,y,z;
                si(x);si(y);si(z);
                update(1,1,n,x,y,z);

            }

            if(tp==1){
                ll x,y;
                si(x);si(y);
                po(query(1,1,n,x,y));
            }

        }
```

```
    }


return 0;
}

//kmp
typedef long long ll;
typedef long l;

#define floop(i,n) for(ll i=0;i<n;i++)
#define floopk(i,n,k) for(ll i=0;i<n;i+=k)
#define si(n) scanf("%ld",&n)
#define po(n) printf("%ld",n)


void fail_fun(char pattern[],l fun[],l n){

    fun[0]=0;fun[1]=0;
    for(l p=2;p<=n;p++){
       l j=fun[p-1];
        for( ; ;){
            if(pattern[j]==pattern[p-1]){
                fun[p]=j+1;
                break;
            }

            if(j==0){
                fun[p]=0;
                break;
            }

            j=fun[j];

        }
    }
}

int main()
{
     //std::ios_base::sync_with_stdio(false);
    l n;
    while(si(n)!=EOF){
            char pattern[n+1];
            l fun[n+1];
            scanf("%s",&pattern);

            fail_fun(pattern,fun,n);

            string text;
            cin>>text;
           // scanf("%s",&text);
            l len=text.length();

            l cnt=0;
            ll i=0;
            ll k=0;
```

```
            for( ; ;){
                    if(i==len)
                        break;

                    if(text[i]==pattern[k]){
                        i++;
                        k++;
                        if(k==n){
                            printf("%lld\n",(i-n));
                            k=fun[n];
                            cnt++;
                        }
                    }
                        else{
                            if(k>0)
                                k=fun[k];
                             else
                                i++;
                        }


            }
            if(cnt==0)
                printf("\n");
    }


return 0;
}


//----------------longest inc subsequence----------------

void bins(int low,int high,int n,int len[])
{
    while(high>low)
    {
        int mid=(high+low)/2;
        if(len[mid]==0 || len[mid]>=n)
        high=mid;
        else
        low=mid+1;
    }
    if(len[high]>n || len[high]==0)
    len[high]=n;
}
int lis(vector<int> arr)
{
    int n= arr.size();
    int len[n+1];
    memset(len,0,sizeof len);
    for(int i=0;i<n;i++)
    {
        bins(1,n,arr[i],len);
    }
    for(int i=n;i>0;i--)
    if(len[i]!=0)
    {
        return i;
    }
```

```
}

//---------------------------------power-------------------------

int powmod(int x, int n)
{
    int w=1,p=x;
    while (n)
    {
        if (n%2) w=w*p%mod;
        p=p*p%mod;
        n/=2;
    }
    return w;
}

//-------------------------------struct sort---------------------------
bool sort_by_x( const data & lhs, const data & rhs )
{
   return lhs.x< rhs.x;
}

//---------------------------------prime sieve-----------------------------
--
long long int primesieve[1000001];
vector <long long int> primes;
void makeprime()
{
    for(long long int i=0;i<=1000000;i++)
    primesieve[i]=1;
    primesieve[1]=0;
    primesieve[0]=0;
    for(long long int i=2;i<=sqrt(1000000);i++)
    {
        if(primesieve[i]==1)
            for(long long int j=i*i;j<=1000000;j=j+i)
            primesieve[j]=0;
    }
    for(long long int i=2;i<1000001;i++)
    if(primesieve[i]==1)
    primes.push_back(i);
}

//---------------------------range prime sieve---------------------------
--------
vector <long long int> primes; //saved prime number upto 1000000
long long int calcpr(long long int l,long long r)
{
    long long int rps[r-l+1];
    for(long long int i=0;i<=r-l;i++)
    rps[i]=1;
    vector<long long int>::iterator it = primes.begin();
    while(it<primes.end())
    {
        long long int x=*it;
        if(x>sqrt(r)+1)
        break;
        long long int k=l/x+(l%x!=0);
        if(k==1)
        k++;
```

```
        k=k*x-l;
        for(long long int j=k;j<=r-l;j=j+x)
        rps[j]=0;
        it++;
    }
    long long int count=0;
    for(long long int i=0;i<=r-l;i++)
    if(rps[i]==1)
    {
        count++;
        //cout<<i+l<<endl;
    }
    return count-(l==1);
}

//---------------------------------dijkstras-----------------------------
-

using namespace std;
typedef vector<int> vi;
typedef pair<int,int> ii;
typedef vector<ii> vii;
typedef vector<vii> vvii;

const int MAX = 1001;
const int MAXINT = 1000000000;

int n;
vvii G(MAX);
vi D(MAX, MAXINT);

void Dijkstra(int s)
{
    set<ii> Q;
    D[s] = 0;
    Q.insert(ii(0,s));

    while(!Q.empty())
    {
        ii top = *Q.begin();
        Q.erase(Q.begin());
        int v = top.second;
        int d = top.first;

        for (vii::const_iterator it = G[v].begin(); it != G[v].end();
it++)
        {
            int v2 = it->first;
            int cost = it->second;
            if (D[v2] > D[v] + cost)
            {
                if (D[v2] != 1000000000)
                {
                    Q.erase(Q.find(ii(D[v2], v2)));
                }
                D[v2] = D[v] + cost;
                Q.insert(ii(D[v2], v2));
            }
        }
    }
```

```
}

int main()
{
    int m, s, t = 0;
    scanf("%d %d %d %d", &n, &m, &s, &t);

    for (int i = 0; i < m; i++)
    {
        int a, b, w = 0;
        scanf("%d %d %d", &a, &b, &w);
        G[a - 1].push_back(ii(b - 1, w));
        G[b - 1].push_back(ii(a - 1, w));
    }

    Dijkstra(s - 1);

    printf("%d\n", D[t - 1]);

    return 0;
}


//--------------fast prime sieve- Sieve of Atkin--------------

using namespace std;
int main (int argc, char* argv[])
{
 //Create the various different variables required
 int limit = 1000000;
 int root = ceil(sqrt(limit));
 bool sieve[limit];
 int primes[(limit/2)+1];
 int insert = 2;
 primes[0] = 2;
 primes[1] = 3;
 for (int z = 0; z < limit; z++)
 sieve[z] = false; //Not all compilers have false as the default boolean
value

 for (int x = 1; x <= root; x++)
 {
    for (int y = 1; y <= root; y++)
    {
    //Main part of Sieve of Atkin
    int n = (4*x*x)+(y*y);
    if (n <= limit && (n % 12 == 1 || n % 12 == 5)) sieve[n] ^= true;
    n = (3*x*x)+(y*y);
    if (n <= limit && n % 12 == 7) sieve[n] ^= true;
    n = (3*x*x)-(y*y);
    if (x > y && n <= limit && n % 12 == 11) sieve[n] ^= true;
    }
 }
 //Mark all multiples of squares as non-prime
 for (int r = 5; r <= root; r++)
 if (sieve[r])
 for (int i = r*r; i < limit; i += r*r)
 sieve[i] = false;

 //Add into prime array
```

```
 for (int a = 5; a < limit; a++)
 {
    if (sieve[a])
    {
        primes[insert] = a;
        insert++;
    }
 }
 return 0;
}

//---------------------prime factors---------------------
// Program to print all prime factors

using namespace std;
vector <int> pf;
void primeFactors(int n)
{
    while (n%2 == 0)
    {
        pf.push_back(2);
        n = n/2;
    }

    for (int i = 3; i <= sqrt(n); i = i+2)
    {
        while (n%i == 0)
        {
            pf.push_back(i);
            n = n/i;
        }
    }
    if (n > 2)
        pf.push_back(n);
}


/* this function calculates (a*b)%c taking into account that a*b might
overflow */
long long mulmod(long long a,long long b,long long c){
    long long x = 0,y=a%c;
    while(b > 0){
        if(b%2 == 1){
            x = (x+y)%c;
        }
        y = (y*2)%c;
        b /= 2;
    }
    return x%c;
}

//---------------------------------BIT functions-------------------------
------------//
int read(int idx){ //reads upto index idx..... note it id 1 indexed BIT
        int sum = 0;
        while (idx > 0){
                sum += tree[idx];
                idx -= (idx & -idx);
        }
        return sum;
```

```
}
//This function updates that particular index value
void update(int idx ,int val){
        while (idx <= MaxVal){
                tree[idx] += val;
                idx += (idx & -idx);
        }
}
//scaling down by factor c
void scale(int c){
        for (int i = 1 ; i <= MaxVal ; i++)
                tree[i] = tree[i] / c;
}

/// ---------------- palindrome ------------ ///
string convert(string s)
{
  int n = s.length();
  if (n == 0) return "^$";
  string ret = "^";
  for (int i = 0; i < n; i++)
    ret += "#" + s.substr(i, 1);

  ret += "#$";
  return ret;
}
int main()
{
    string s;
    while(cin>>s)
    {
        string T = convert(s);
        int n = T.length();
        int P[n];
        int C = 0, R = 0;
        for (int i = 1; i < n-1; i++)
        {
            int i_mirror = 2*C-i; // equals to i' = C - (i-C)

            P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;

            // Attempt to expand palindrome centered at i
            while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
              P[i]++;

            // If palindrome centered at i expand past R,
            // adjust center based on expanded palindrome.
            if (i + P[i] > R)
            {
              C = i;
              R = i + P[i];
            }
        }
        int maxLen = 0;
        int centerIndex = 0;
        for(int i=n-3;i>0;i--)
        {
            //cout<<P[i]<<" ";
            if(P[i]==n-i-2)
            {
```

```
                maxLen = P[i];
                centerIndex = i;
            }
        }
        //cout<<endl;
        string rev=s.substr(0,s.length()-maxLen);
        reverse(rev.begin(),rev.end());
        cout<<s.substr(0,s.length()-maxLen)<<s.substr((centerIndex - 1 -
maxLen)/2, maxLen)<<rev<<endl;
    }
}
///- ------------------z algo----------------------//
void zalgo(int z[],string s)
{
    int r=-1,l=-1;
    for(int i=1;i<n;i++)
    {
        if(r<i)
        {
            for(int j=i;j<n;j++)
            if(s[j]==s[j-i])
                z[i]++;
            else
                break;
        }
        else
        {
            int x=i-l;
            z[i]=z[x];
            if(z[x]>=r-i+1)
            {
                z[i]=r-i+1;
                for(int j=z[i],k=r+1;k<n;j++,k++)
                {
                    if(s[j]==s[k])
                        z[i]++;
                    else
                        break;
                }
            }
        }
        if(r<i+z[i]-1)
            l=i,r=i+z[i]-1;
        //cout<<z[i]<<" "<<r<<" "<<l<<endl;
    }
}
//------------ DSU ---------------//
int parent[100001],rank[100001];
int find(int x)
{
    if(parent[x]==x)
        return x;
    parent[x]=find(parent[x]);
    return parent[x];
}
int merge(int u,int v)
{
    int pu=find(u);
    int pv=find(v);
    if(pu==pv)
```

```
        return 0;
    if(rank[pu]>rank[pv])
    {
        parent[pv]=pu;

    }
    else if(rank[pu]<rank[pv])
    {
        parent[pu]=pv;

    }
    else
    {
        parent[pu]=pv;

        rank[pv]++;
    }
    return 1;
}
//------------- euler function ----------///
long long euler(long long k)
{
    long long r = k,p=2;
    while(p*p<=k)
    {

        if(k%p==0)
        r=r-r/p;
        while(k%p==0)
            k=k/p;
        p++;
    }
    if(k>1)
        r=r-r/k;
    return r;
}
//------------//
j=s[i].find("miao.");
k=s[i].rfind("lala.");

//------strongly connected components---------//

int n,m,ti,counter,value;
int cost[100001];
int visit[100001];
struct data
{
    int v,nn;
    vector <int> neb;
}node1[100001],node2[100001];

struct data1
{
    int index,tim;
}arr[100001];

bool sortbytime(const data1 & lhs,const data1 & rhs)
{
   return lhs.tim> rhs.tim;
}
```

```
void dfs(int ind)
{
    node1[ind].v=1;
    for(int i=0;i<node1[ind].nn;i++)
    if(node1[node1[ind].neb[i]].v==0)
    dfs(node1[ind].neb[i]);
    ti=ti+1;
    arr[ind].tim=ti;
}

void dfs1(int ind)
{
    node2[ind].v=1;
    if(cost[ind]<value)
    {
        value=cost[ind];
        counter=1;
    }
    else if(cost[ind]==value)
    {
        counter++;
    }
    for(int i=0;i<node2[ind].nn;i++)
    if(node2[node2[ind].neb[i]].v==0)
    dfs1(node2[ind].neb[i]);
}

int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>cost[i];
    cin>>m;
    for(int i=0;i<m;i++)
    {
        int ui,vi;
        cin>>ui>>vi;
        ui--;
        vi--;
        node1[ui].neb.push_back(vi);
        node1[ui].nn++;
        node2[vi].neb.push_back(ui);
        node2[vi].nn++;
    }
    for(int i=0;i<n;i++)
    if(node1[i].v==0)
    dfs(i);

    for(int i=0;i<n;i++)
        arr[i].index=i;
    sort(arr,arr+n,sortbytime);
    long long ans1=0,ans2=1;
    //cout<<node2[0].nn<<" "<<node2[1].nn<<" "<<node2[2].nn<<endl;
    for(int i=0;i<n;i++)
    {
        //cout<<arr[i].index<<" "<<arr[i].tim<<endl;
        if(node2[arr[i].index].v==0)
        {
            counter=0;
```

```
            value=1000000001;
            dfs1(arr[i].index);
            //cout<<value<<" "<<counter<<endl;
            ans1=ans1+value;
            ans2=(ans2*counter)%1000000007;
        }
    }
    cout<<ans1<<" "<<ans2<<endl;

}


//------------------stable marige problem------------------//
int men[501][501];
int wom[501][501];
int ans[501];
int start[501];
int n;
vector <int> s;


int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        cin>>n;
        memset(ans,-1,sizeof(ans));
        memset(start,0,sizeof(start));
        s.clear();
        for(int i=0;i<n;i++)
        {
            int x;
            cin>>x;
            x--;
            for(int j=0;j<n;j++)
            {
                cin>>wom[x][j],wom[x][j]--;
            }
        }
        for(int i=0;i<n;i++)
        {
            int x;
            cin>>x;
            x--;
            for(int j=0;j<n;j++)
            {
                cin>>men[x][j],men[x][j]--;
            }
        }
        for(int i=0;i<n;i++)
            s.push_back(i);
        //cout<<"-----------"<<endl;
        while(s.size()>0)
        {
            int m=s[0];
            int w=men[m][start[m]];
            //cout<<m<<" "<<w<<endl;
            if(ans[w]==-1)
```

```
            {
                ans[w]=m;
                s.erase (s.begin());
                start[m]++;
            }
            else
            {
                int m1=ans[w];
                for(int i=0;i<n;i++)
                {
                    if(wom[w][i]==m1)
                    {
                        start[m]++;
                        break;
                    }
                    else if(wom[w][i]==m)
                    {
                        ans[w]=m;
                        s.erase (s.begin());
                        start[m]++;
                        s.push_back(m1);
                        break;
                    }
                }
            }
        }
        int final[n];
        for(int i=0;i<n;i++)
            final[ans[i]]=i;
        for(int i=0;i<n;i++)
            cout<<i+1<<" "<<final[i]+1<<endl;
    }
}
////////////////////////////////////////////////////////
//josepheus problem
int josephus(int n, int k)
{
  if (n == 1)
    return 1;
  else
    /* The position returned by josephus(n - 1, k) is adjusted because
the
       recursive call josephus(n - 1, k) considers the original position
       k%n + 1 as position 1 */
    return (josephus(n - 1, k) + k-1) % n + 1;
}
// return ( ((josephus(n-1, k) +k-1) % n +1) + x-1) % n + 1
// return the above if one needs to start from anywhere else of the
// Driver Program to test above function
int main()
{
  int n = 14;
  int k = 2;
  printf("The chosen place is %d", josephus(n, k));
  return 0;
}
// josepheus iterative
////////////////////////////////////////////////
#include <stdio.h>
```

```
int solve(int n, int d)
{
    register int i, a = 0;
    for(i=2; i<=n; i++)
        a = (a+d)%i;
    return a;
}

int main()
{
    int n, d;
    while(scanf("%d%d", &n, &d)==2 && n+d)
        printf("%d %d %d\n", n, d, solve(n, d) + 1);
    return 0;
}
//// gcd ////
template <typename T>
T gcd(T a, T b)
{
    T k;
    while(b)
    {
        a %= b;
        k = a;
        a = b;
        b = k;
    }
    return a;
}
////////////// nCr precompute ///////////////////

#define MOD 1000000007

using namespace std;

long long C[2222][2222],a[1111],n,sum=0,ans=1;

void pre_comp() {
  C[0][0]=1;
  for(int i=1;i<=2001;++i) {
    C[i][0]=1;
    for(int k=1;k<=i;++k) {
      C[i][k]=C[i-1][k]+C[i-1][k-1];
      C[i][k]=C[i][k]%MOD;
    }
  }
}

int main() {

  pre_comp();
  cin>>n;
  for(int i=0;i<n;++i) cin>>a[n-i-1];
  for(int i=n-1;i>=0;i--) {
    ans=(1LL*ans*C[sum+a[i]-1][a[i]-1])%MOD;
    sum+=a[i];
  }
  cout<<ans;
}
```

```
///////////////// dsu ///////////////////
int parent[100];
int rank[100];

void createSet(int x){
  parent[x] = x;
  rank[x] = 0;
}

int findSet(int x){
  if(x != parent[x]) parent[x] = findSet(parent[x]) ;
  return parent[x];
}

void unionSet(int x, int y){
  int parentX = findSet(x);
  int parentY = findSet(y);
  if ( rank[parentX] > rank[parentY] ) parent[parentY] = parentX;
  else parent[parentX] = parentY;
  if ( rank[parentX] == rank[parentY]) rank[parentY] +=1;
}


int main(){
  int n;
  cin>>n;
  for(int i=0;i<100;i++) {
      createSet(i);
  }
  //assuming there must be n pairs of numbers

  for(int i=0;i<n; i++){
    int a, b;
    cin>> a >> b;
    if(findSet(a)!=findSet(b))
      unionSet(a,b);
  }

  for(int i=1;i<=n;i++) cout<<findSet(i)<<endl;

}

////////////////////////////dijkstra ////////////////////////////////

#define ALL(p) p.begin(),p.end()
#define INF 2147483647
#define pb(x) push_back(x)
#define pii pair< int , int >
#define MAX 100010

vector < pii > G[MAX];
int d[MAX], f[MAX];

int process (int n, int e)
{
  int start, end;
  int u,v ,w,we,wn,size;
  for(int i=0;i<n+1;i++) /*Initialize Graph to 0*/
  {
    G[i].clear();
```

```
    d[i] = INF;   /* d is array of best estimates of cost*/
    f[i] = 0;   /* f is array of (previous) predeseccors of the vertex*/
  }
  // return 1;

  for(int i=0;i<e;i++)
  {
    scanf("%d %d %d", &u,&v, &w);
    G[u].pb(make_pair(w,v));
    G[v].pb(make_pair(w,u));
  }
  // return 1;
  cin>>start>>end;
  d[start]= 0; //trivial case : the distance from source to source
  priority_queue < pii , vector < pii > , greater < pii > > Q;
  // template <object_type, container_type, comparator_function>
  Q.push(pii(0, start));

  // return 1;
  while(!Q.empty())
  {
    u = Q.top().second;
    wn = Q.top().first;
    Q.pop();
    size = G[u].size();
    for(int i=0;i<size;i++)
    {
      v = G[u][i].second;
      we = G[u][i].first;
      if(!f[v] && wn+we<=d[v])
      {
        d[v]=wn+we;
        Q.push(pii(d[v],v));

      }
    }
    f[u]= 1;
    if(u==end) break;
  }
  return d[end];
}

int main() {
  int t, n, e, dist;
  scanf("%d", &t);
  while(t--) {
    scanf("%d %d", &n, &e);
    dist = process(n, e);
    if(dist==INF) printf("NO\n");
    else printf("%d\n", dist);
  }
  return 0;
}
////////////////////////// recursive number gen //////////////////
long long n,s;
void p(long long x,int a7,int a4)
{
  if ( x>=n&&a7+1==a4&&(x<s||!s))s=x;
  if ( x<n*100 )
  {
```

```
    p(x*10+4,a7,a4+1);
    p(x*10+7,a7+1,a4);
  }
}
int main()
{
  cin>>n;
  p(0,0,0);
  if ( n==0 )  s=47;
  cout<<s<<endl;
}


//------------------------------------------------------------------//
////////////////////////////////////////////////////////////////////
//////////////////////////////

-----------Polygon Formulas----------------------



(N = # of sides and S = length from center to a corner)

Area of a regular polygon = (1/2) N sin(360Â°/N) S2
Sum of the interior angles of a polygon = (N - 2) x 180Â°

The number of diagonals in a polygon = 1/2 N(N-3)
The number of triangles (when you draw all the diagonals from one vertex)
in a polygon = (N - 2)
---------------Triangle formula-------------------
area =1/2*ab sinC
area = (s(sâ^'a)(sâ^'b)(sâ^'c))^1/2
---------modulor ----------------------------------
 a^-1 mod p= a^p-2 mod p
(a - b) mod p = ((a mod p - b mod p) + p) mod p
(a / b) mod p = ((a mod p) * (b^(-1) mod p)) mod p

------------------GCD LCM---------------------------------
For integers N1, ..., Nk, k â‰¥ 2,
lcm(gcd(N1, M), gcd(N2, M), ..., gcd(Nk, M)) = gcd(lcm(N1, ..., Nk), M)
gcd(lcm(N1, M), lcm(N2, M), ..., lcm(Nk, M)) = lcm(gcd(N1, ..., Nk), M)

gcd(PÂ·N, PÂ·M) = PÂ·gcd(N, M)
lcm(PÂ·N, PÂ·M) = PÂ·lcm(N, M).
---------------prime test---------

A simple, but very inefficient primality test uses Wilson's theorem,
which states that p is prime if and only if:
(p-1)(mod p) != -1(mod p)



------------basic bfs---------------------
bool bfs(int s)
{
    int i, u, v, sz;
    queue< int > Q;
    Q.push(s);
    flag[s] = 1;
```

```
    while(!Q.empty())
    {
        u = Q.front();
        Q.pop();
        sz = G[u].size();
        for(i=0; i<sz; i++)
        {
            v = G[u][i];
            if(pre[u] != v && flag[v]) return false;
            else if(!flag[v])
            {
                pre[v] = u;
                flag[v] = 1;
                Q.push(v);
            }
        }
    }
    for(i=1; i<=N; i++)
        if(!flag[i])
            return false;
    return true;
}


// /**/
-----------------basic dfs ------------
#include <cstdio>
#include <vector>
#include <queue>
#include <bits/stdc++.h>
using namespace std;

#define MAX 10001

int N, E;
vector< int > G[MAX];
bool flag[MAX];
int pre[MAX];

bool dfs(int s)
{
    int i, u, v, sz;
    stack< int > stk;
    stk.push(s);
    flag[s] = 1;
    while(!stk.empty())
    {
        u = stk.top();
        stk.pop();
        sz = G[u].size();
        for(i=0; i<sz; i++)
        {
            v = G[u][i];
            if(pre[u] != v && flag[v]) return false;
            else if(!flag[v])
            {
                pre[v] = u;
                flag[v] = 1;
                stk.push(v);
            }
```

```
        }
    }
    for(i=1; i<=N; i++)
        if(!flag[i])
            return false;
    return true;
}

int main()
{
    int i, u, v, s;
    scanf("%d %d", &N, &E);
    for(i=0; i<E; i++)
    {
        scanf("%d %d", &u, &v);
        //scanning two nodes which are the edges in the graph G
        s = u;
        G[u].push_back(v);

        G[v].push_back(u);
    }
    if(E!=N-1) //checking for the basic condition of a graph
        printf("NO\n");
    else
    {
        if(dfs(s)) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
-- ----- --

//suffix arrays
typedef long long ll;
typedef long l;
#define floop(i,n) for(ll i=0;i<n;i++)
#define floopk(i,n,k) for(ll i=0;i<n;i+=k)
#define si(n) scanf("%ld",&n)
#define po(n) printf("%ld",n)
struct node{

    l rank0;
    l rank1;
    l index;

};
l cmp( const node &n1,const  node &n2){

    return (n1.rank0==n2.rank0)?(n1.rank1<n2.rank1):(n1.rank0<n2.rank0);
}
l create_suffixarray(node suffixarray[],l n,char text[]){

        for(l p=0;p<n;p++){
            suffixarray[p].rank0=text[p]-'a';
            suffixarray[p].rank1=(p+1<n)?(text[p+1]-'a'):400000;
            suffixarray[p].index=p;
        }

        sort(suffixarray,suffixarray+n,cmp);
```

```
        l ind[100005];

        for( l k=4;k<2*n;k*=2){

                l rnk=0;
                l prev=suffixarray[0].rank0;
                suffixarray[0].rank0=rnk;
                ind[suffixarray[0].index]=0;

                for( l p=1;p<n;p++){

                        if(suffixarray[p].rank0==prev &&
suffixarray[p].rank1==suffixarray[p-1].rank1){
                           // prev = suffixarray[p].rank0;
                            suffixarray[p].rank0=rnk;
                        }

                        else{
                             prev=suffixarray[p].rank0;
                             suffixarray[p].rank0=++rnk;
                        }

                        ind[suffixarray[p].index]=p;



                }

                for( l v=0;v<n;v++){

                    l nextind=(suffixarray[v].index)+(k/2);
                    //nextind%=n;

suffixarray[v].rank1=((nextind<n)?(suffixarray[ind[nextind]].rank0):40000
0);

                }

                sort(suffixarray,suffixarray+n,cmp);


        }

      return suffixarray[0].index;



}

int main()
{
   //std::ios_base::sync_with_stdio(false);
  l t;
  si(t);
  while(t--){

            char ch[10005],ch1[30005];
            node suffixarray[30005];
            //scanf("%s",&ch);
            cin>>ch;
```

```
            l n=strlen(ch);
            strcpy(ch1,ch);
            strcat(ch1,ch);


            //ll ans=(n*(n+1))/2;
            //ll ans=0;
            l ans= create_suffixarray(suffixarray,2*n,ch1);
            ans++;
            cout<<ans<<"\n";



  }


return 0;
}

//
//Implementation of Dijkstra's algorithm using adjacency lists
// and priority queue for efficiency.
//
// Running time: O(|E| log |V|)

#include <queue>
#include <stdio.h>

using namespace std;
const int INF = 2000000000;
typedef pair<int,int> PII;

int main(){

  int N, s, t;
  scanf ("%d%d%d", &N, &s, &t);
  vector<vector<PII> > edges(N);
  for (int i = 0; i < N; i++){
    int M;
    scanf ("%d", &M);
    for (int j = 0; j < M; j++){
      int vertex, dist;
      scanf ("%d%d", &vertex, &dist);
      edges[i].push_back (make_pair (dist, vertex)); // note order of
arguments here
    }
  }

  // use priority queue in which top element has the "smallest" priority
  priority_queue<PII, vector<PII>, greater<PII> > Q;
  vector<int> dist(N, INF), dad(N, -1);
  Q.push (make_pair (0, s));
  dist[s] = 0;
  while (!Q.empty()){
    PII p = Q.top();
    if (p.second == t) break;
    Q.pop();

    int here = p.second;
```

```
    for (vector<PII>::iterator it=edges[here].begin();
it!=edges[here].end(); it++){
      if (dist[here] + it->first < dist[it->second]){
        dist[it->second] = dist[here] + it->first;
        dad[it->second] = here;
        Q.push (make_pair (dist[it->second], it->second));
      }
    }
  }

  printf ("%d\n", dist[t]);
  if (dist[t] < INF)
    for(int i=t;i!=-1;i=dad[i])
      printf ("%d%c", i, (i==s?'\n':' '));

  return 0;
}
//SCC.cc 19/35 strongly connected components

#include<memory.h>
struct edge{int e, nxt;};
int V, E;
edge e[MAXE], er[MAXE];
int sp[MAXV], spr[MAXV];
int group_cnt, group_num[MAXV];
bool v[MAXV];
int stk[MAXV];
void fill_forward(int x)
{
  int i;
  v[x]=true;
  for(i=sp[x];i;i=e[i].nxt) if(!v[e[i].e]) fill_forward(e[i].e);
  stk[++stk[0]]=x;
}
void fill_backward(int x)
{
  int i;
  v[x]=false;
  group_num[x]=group_cnt;
  for(i=spr[x];i;i=er[i].nxt) if(v[er[i].e]) fill_backward(er[i].e);
}
void add_edge(int v1, int v2) //add edge v1->v2
{
  e [++E].e=v2; e [E].nxt=sp [v1]; sp [v1]=E;
  er[  E].e=v1; er[E].nxt=spr[v2]; spr[v2]=E;
}
void SCC()
{
  int i;
  stk[0]=0;
  memset(v, false, sizeof(v));
  for(i=1;i<=V;i++) if(!v[i]) fill_forward(i);
  group_cnt=0;
  for(i=stk[0];i>=1;i--) if(v[stk[i]]){group_cnt++;
fill_backward(stk[i]);}
}

//EulerianPath.cc 20/35

struct Edge;
```

```
typedef list<Edge>::iterator iter;

struct Edge
{
  int next_vertex;
  iter reverse_edge;

  Edge(int next_vertex)
    :next_vertex(next_vertex)
    { }
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];   // adjacency list

vector<int> path;

void find_path(int v)
{
  while(adj[v].size() > 0)
  {
    int vn = adj[v].front().next_vertex;
    adj[vn].erase(adj[v].front().reverse_edge);
    adj[v].pop_front();
    find_path(vn);
  }
  path.push_back(v);
}

void add_edge(int a, int b)
{
  adj[a].push_front(Edge(b));
  iter ita = adj[a].begin();
  adj[b].push_front(Edge(a));
  iter itb = adj[b].begin();
  ita->reverse_edge = itb;
  itb->reverse_edge = ita;
}


// simple stl implementations
//map

int main()
{
   map<int, string> Employees;

   // 1) Assignment using array index notation
   Employees[5234] = "Mike C.";
   Employees[3374] = "Charlie M.";
   Employees[1923] = "David D.";
   Employees[7582] = "John A.";
   Employees[5328] = "Peter Q.";

   cout << "Employees[3374]=" << Employees[3374] << endl << endl;

   cout << "Map size: " << Employees.size() << endl;
```

```
   for( map<int,string>::iterator ii=Employees.begin();
ii!=Employees.end(); ++ii)
   {
       cout << (*ii).first << ": " << (*ii).second << endl;
   }
}

//2
#include <string.h>
#include <iostream>
#include <map>
#include <utility>

using namespace std;

int main()
{
   map<int, string> Employees;

   // 1) Assignment using array index notation
   Employees[5234] = "Mike C.";
   Employees[3374] = "Charlie M.";
   Employees[1923] = "David D.";
   Employees[7582] = "John A.";
   Employees[5328] = "Peter Q.";

   cout << "Employees[3374]=" << Employees[3374] << endl << endl;

   cout << "Map size: " << Employees.size() << endl;

   for( map<int,string>::iterator ii=Employees.begin();
ii!=Employees.end(); ++ii)
   {
       cout << (*ii).first << ": " << (*ii).second << endl;
   }
}

//find
vector<int> v;
  // the next line creates 'it'. Its type is vector<int>::iterator
  // which is verbose, but logical - it's an iterator for dealing with
vector<int>
  // Not that :: is used because the thing before it is a class, not an
object.
  vector<int>::iterator it;
  ...
  it = find (v.begin(), v.end(), 30);
  if (it ==  v.end())
    cout << "30 not found " << endl;
  else
    cout << "30 is in v " << endl;

//countcount - finds how many elements match a given value
  vector<int> v;
  ...
  cout << "30 is in v "  << count(v.begin(), v.end(), 30) << " times" <<
endl;

//count_if - finds how many elements match a given condition
```

```
// The next function returns true if n is a multiple of 3 -
// the % operator calculates the remainder after division
bool multipleOf3(int n) {
  return not (n%3);
}

int main() {
  vector<int> v(10);
  for (int i=0;i<v.size();i++)
    v[i]=i;
  cout << count_if(v.begin(), v.end(),multipleOf3) << " multiples" <<
endl;
}

//for_each - like a "for" loop, but can't change the container's values
    void square(int n) {
       cout << n*n << endl;
    }

    int main() {
    vector<int> v;
    ...
    // print the square of all the values in v
    for_each(v.begin(), v.end(), square);
    }

//transform - like a "for" loop; you can change the container's values
    int square(int n) {
       return  n*n;
    }

    int main() {
    vector<int> v;
    ...
    // replace each value in v by its square
    transform(v.begin(), v.end(), square);
    }
//replace - replaces values
  vector<int> v;
    ...
  // replace 7s by 3s
  replace(v.begin(), v.end(), 7,3);

//fill - fills a container with a value
    vector<int> v;
    fill(v,9);

//copy
    int main() {
  vector<int> v(10);
  for (int i=0;i<v.size();i++)
    v[i]=i;
  list<int> l(10);
  copy(v.begin(), v.end(),l.begin());
}
//remove - removes all the items with a particular value (actually it
moves all those items to the end of the vector - you can delete them
later if you wish)
   vector<int> v;
   vector<int>::iterator it;
```

```
   it=remove (v.begin(), v.end(), 7);

//unique - puts duplicates at the end of the vector (but copies need to
be contiguous with originals). You can delete the duplicates afterwards.
// full program


int main () {
string s="holly  root";
 string::iterator it;
 cout << "s original:    " << s<<endl;
 it=unique(s.begin(),s.end()); // 'it' points to the 1st of the
duplicates
 cout << "s after unique:" << s<<endl;
 s.erase(it, s.end());
 cout << "s after erase: " << s<<endl;
}
//unique_copy - copies the unique elements to a new place (but copies
need to be contiguous with originals).
// full progra

int main () {
string s="holly  root";
string t="           ";
 cout << "s original:    " << s<<endl;
 // copy the unique characters from string s to string t (but t needs to
be
 // big enough to contain the characters)
 unique_copy(s.begin(),s.end(),t.begin());
 cout << "s after: " << s<<endl;
 cout << "t after: " << t<<endl;
}
//set_intersection - intersection of 2 sorted containers
// full program
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main () {
  vector<int> set1(3);
  vector<int> set2(3);
  vector<int> answer(3);
  set1[0]=1;  set1[1]=2;  set1[0]=3;
  set2[0]=2;  set2[1]=4;  set2[0]=6;
  vector<int>::iterator it;
  // the next line puts the common values into the 'answer' vector
  // returning a pointer to the element just beyond answer's last element
  it=set_intersection (set1.begin(), set1.end(),set2.begin(), set2.end(),
                       answer.begin());
   cout  << "Number of common values="  << answer.size() << endl;
min_element - finds the minimum
   vector<int> v;
   vector<int>::iterator it;
   it=min_element(v.begin(), v.end());
   cout  << *it << endl;
next_permutation - finds the next permutation
// full program
#include <iostream>
#include <algorithm>
```

```
using namespace std;

int main () {
  vector<int> myints(3);
   myints[0]=1;  myints[1]=2;  myints[2]=3;
  sort (v.begin(),v.end()); // only really needed if the elements are
unsorted
  cout << "The 3! possible permutations with 3 elements:\n";

  do {
    cout << myints[0] << " " << myints[1] << " " << myints[2] << endl;
  } while ( next_permutation (v.begin(),v.end()) );
}
accumulate - adds the elements (needs #include <numeric>)
partial_sum - replaces each element by successive partial sums (needs
#include <numeric>)
// full program
#include <iostream>
#include <algorithm>
#include <numeric>
using namespace std;

int main () {
// create a vector big enough to hold 10 ints
  vector<int> v(10);
// fill it with 1s
  fill(v.begin(), v.end(), 1);
// work out partial sums and put the answer back in v
  partial_sum(v.begin(), v.end(),v.begin() );
// v now contains 1....10. Now sum the elements using 0 as the initial
sum
  cout << "Sum of elements=" << accumulate(v.begin(), v.end(),0)  <<
endl;
}

//multimap
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main ()
{
  multimap<string,string> car;

  car.insert(pair<string,string>("Smith","Ford"));
  car.insert(pair<string,string>("Jones","Jaguar"));
  car.insert(pair<string,string>("Smith", "Nissan"));
  cout << "Smith has " << car.count("Smith") << " cars" <<  endl;
}


#include <iostream>
#include <map>
#include <string>
using namespace std;

int main ()
{
  multimap<string,string> car;
```

```
  multimap<string,string>::iterator it;
  pair<multimap<string,string>::iterator,multimap<string,string>::
iterator> two_its;
  car.insert(pair<string,string>("Smith","Ford"));
  car.insert(pair<string,string>("Jones","Jaguar"));
  car.insert(pair<string,string>("Smith", "Nissan"));

  two_its=car.equal_range("Smith");
  cout  << "Smith owns";
  for (it=two_its.first; it!=two_its.second; ++it)
      cout  << " "  << (*it).second;

  cout  <<  endl;
}
//map --sample

#include <iostream>
#include <set>
#include <vector>
#include <algorithm>



using namespace std;

typedef multiset<int, greater<int> >::iterator it_type;

int main(){
  int N,B,SG,SB;
  int i,temp;
  int actualB;
  multiset<int, greater<int> > SG_army;
  multiset<int, greater<int> > SB_army;
  it_type my_iterator;

  vector<int> fightersSG;
  vector<int> fightersSB;

  bool flag_first = false;

  cin>>N;
  while(N--){
    if(flag_first){
      cout<<endl;
    }
    cin>>B>>SG>>SB;

    for(i=0; i<SG; i++){
      cin>>temp;
      SG_army.insert(temp);
    }

    for(i=0; i<SB; i++){
      cin>>temp;
      SB_army.insert(temp);
    }

    while(!SG_army.empty() && !SB_army.empty()){
      actualB = min(B, min((int)SG_army.size(),(int)SB_army.size()));
```

```
      i=0;

      for(my_iterator = SG_army.begin(); i<actualB; i++){
        fightersSG.push_back(*my_iterator);
        SG_army.erase(my_iterator++);
      }

      i=0;
      for(my_iterator = SB_army.begin(); i<actualB; i++){
        fightersSB.push_back(*my_iterator);
        SB_army.erase(my_iterator++);
      }

      for(i=0;i<actualB;i++){
        if(fightersSB[i] >= fightersSG[i]){
          fightersSB[i] -= fightersSG[i];
          fightersSG[i] = 0;
        }else{
          fightersSG[i] -= fightersSB[i];
          fightersSB[i] = 0;
        }
      }

      for(i=0;i<actualB;i++){
        if(fightersSB[i]!=0){
          SB_army.insert(fightersSB[i]);
        }
        if(fightersSG[i]!=0){
          SG_army.insert(fightersSG[i]);
        }
      }

      fightersSG.clear();
      fightersSB.clear();
    }

    if(SG_army.empty() && SB_army.empty()){
      cout<<"green and blue died"<<endl;
    }else if(!SG_army.empty()){
      cout<<"green wins"<<endl;
      for(my_iterator = SG_army.begin(); my_iterator!= SG_army.end();
my_iterator++){
        cout<<*my_iterator<<endl;
      }
    }else{
      cout<<"blue wins"<<endl;
      for(my_iterator = SB_army.begin(); my_iterator!= SB_army.end();
my_iterator++){
        cout<<*my_iterator<<endl;
      }
    }
    flag_first = true;

    SG_army.clear();
    SB_army.clear();
  }
  return 0;
}

//set
```

set

One of the most fundamental ways to store information is to have a set of objects. Defining a set of integers is done with

```
set<int> a;
```

To add, remove and check for single elements in a set:

```
a.insert(7);         // Insert integer 7 in the set
a.erase(5);          // Remove integer 5 (if it exist) from the set
if (a.find(7)!=a.end())
  ; // Integer 7 exists in the set
else
  ; // Integer 7 does not exist
cout << a.size() << endl; // Print the number of elements in the set a
```

Other common set operations includes finding the union, intersection and difference between two sets.

```
set<int> a,b,un,in,di;
..
..
set_union(a.begin(),a.end(),b.begin(),b.end(),insert_iterator<set<int>
>(un,un.begin()));
set_intersection(a.begin(),a.end(),b.begin(),b.end(),insert_iterator<set<
int> >(in,in.begin()));
set_difference(a.begin(),a.end(),b.begin(),b.end(),insert_iterator<set<in
t> >(di,di.begin()));

//iterators
set<int> a;
map<string,string> b;
..
..
for(set<int>::iterator i=a.begin();i!=a.end();++i)
  cout << *i << endl;
for(map<string,string>::iterator i=b.begin();i!=b.end();++i)
  cout << i->first << " => " << i->second << endl;

/*

TASK: BLINNET
ALGO: kruskal*/

typedef pair< int, int > pii;
typedef pair< int, pii > edge;
const int MAX = 10001;

vector< edge > edges;
int parent[MAX];

int find(int u) {
  if(u != parent[u]) parent[u] = find(parent[u]);
  return parent[u];
}

int main() {
  int t, i, n, e, u, v, w, cost, sz;
  char dump[20];
  //freopen("in.txt", "r", stdin);
  //freopen("out.txt", "w", stdout);
```

```
  scanf("%d", &t);
  while(t--) {
    edges.clear();
    scanf("%d", &n);
    for(i = 1; i <= n; i++) {
      parent[i] = i;
      scanf("%s%d", dump, &e);
      while(e--) {
        scanf("%d%d", &v, &w);
        if(v > i) edges.push_back(edge(w, pii(i, v)));
      }
    }
    sort(edges.begin(), edges.end());
    sz = edges.size();
    for(cost=i=0; i < sz; i++) {
      u = find(edges[i].second.first);
      v = find(edges[i].second.second);
      w = edges[i].first;
      if(u != v) {
        cost += w;
        parent[u] = parent[v];
      }
    }
    printf("%d\n", cost);
  }
  return 0;
}
```